PHP-generics

Сухачёв Антон

cdnnow!



- Что такое дженерики и зачем они?
- Какие есть подходы к реализации дженериков?
- Как сделать PHP-дженерики на PHP?



Сильная/слабая типизация

Сильная (Java)

```
var foo = 0;
foo = "string";// ошибка

if("string" == true) {// ошибка
    System.out.print("ok");
}
```

Слабая (PHP)

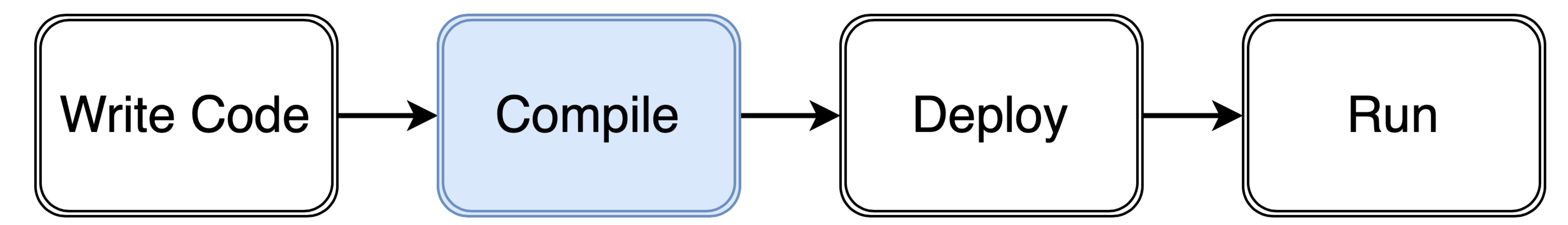
```
$foo = 0;
$foo = 'string';// изменение типа

if('string'=== true) {//преобразование
    echo 'ok';
}
```

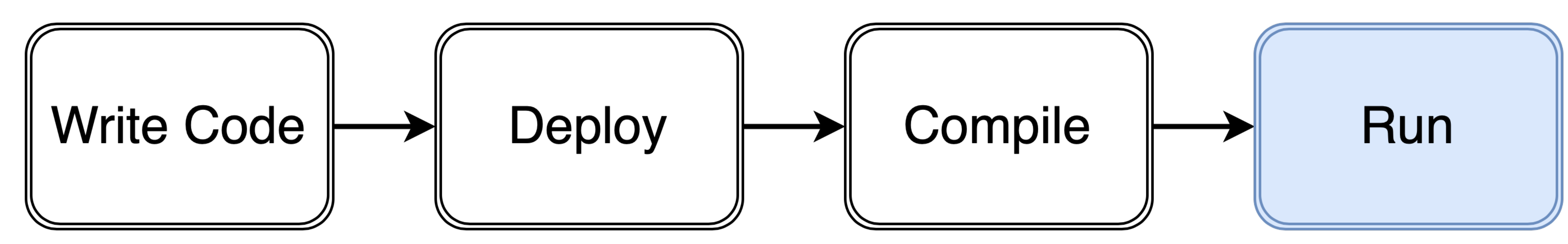



Статическая/динамическая типизация

Статическая (Java)



Динамическая (PHP)



Смешанная типизация (PHP)

```php
<?php

function test(): void {// ошибка компиляции
    return 1;
}
```




PHP – динамический слаботипизированный



Проверки типов на уровне языка
Писать/читать код проще

```php
<?php

// developer
function pay($account, $amount) {}

// user
$receipt = pay($account, $amount);
```




Проверки типов на уровне языка
Писать/читать код проще

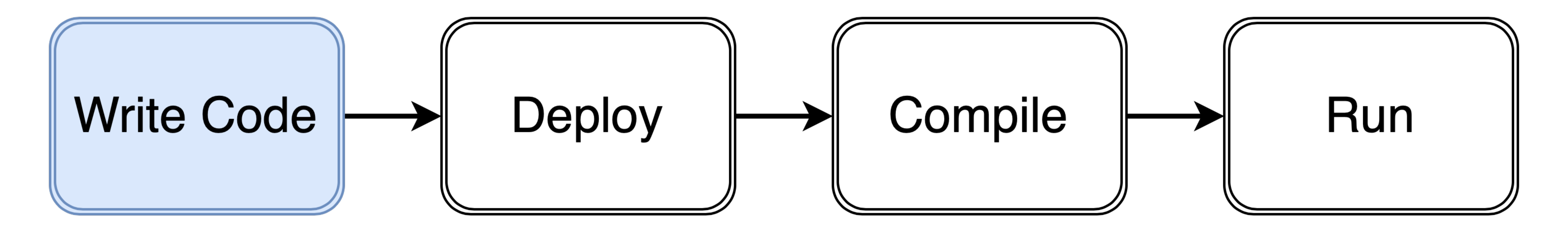
```php
<?php

// developer
function pay(Account $account, int $amount): Receipt {}

// user
$receipt = pay($account, $amount);
```




Статический анализ кода (Psalm/PHPStan/Phan/IDE)





Аннотации

```php
<?php

/**
 * @param Account $account
 * @param int $amount
 * @return Receipt
 */
function pay($account, $amount) {}
```




Зачем нужна типизация в PHP?

- проверки типов на уровне языка
- писать код проще
- читать код проще
- статический анализ
- аннотации могут устаревать



Где можно использовать типы?

```php
<?php

class UserService
{
    private Repository $repository;

    public function create(string $name): User {}
}
```




Коллекции

```php
<?php

class UserCollection
{
    /**
     * @param User[] $users
     */
    public function set(array $users): void {}
}
```




Коллекции

```php
<?php

class UserCollection {
    public function add(User $user): void {}
}
```




Коллекции

- UserCollection
- DogCollection
- CatCollection
- BirdCollection



Дженерики

```php
<?php

class Collection<T> {
    public function add(T $value): void {}
}
```




Дженерики

```php
<?php

$userCollection = Collection<User>();
$userCollection->add(new User());

$dogCollection = Collection<Dog>();
$dogCollection->add(new Dog());

$catCollection = Collection<Cat>();
$catCollection->add(new Cat());

$birdCollection = Collection<Bird>();
$birdCollection->add(new Bird());
```




Подходы к реализации дженериков

- type erasure
- reification
- monomorphization



Type erasure

До

```php
<?php

class Container<T> {

    private T $data;

    public function __construct(T $data){}
}

$container = new Container<int>(1);
```

После

```php
<?php

class Container {

    private $data;

    public function __construct($data){}
}

$container = new Container(1);
```




Type erasure

До

```php
<?php

class Container<T> {

    public function foo($data) {

        T::staticFunction();

        $object = new T();

        if($data instanceof T) {

        }
    }
}
```

После

```php
<?php

class Container {

    public function foo($data) {

        ::staticFunction();

        $object = new ();

        if($data instanceof ) {

        }
    }
}
```




Type erasure

+ простая реализация

+ проверка типов на стадии статического анализа или компиляции

- ограниченное использование дженериков в коде

- нет типов дженериков в runtime (reflection)



Monomorphization

До

```php
<?php

class Container<T> {

    private T $data;

    public function __construct(T $data){
        $this->data = $data;
    }
}

$intContainer = new Container<int>(1);
```

После

```php
<?php

class ContainerForInt {

    private int $data;

    public function __construct(int $data){
        $this->data = $data;
    }
}

$intContainer = new ContainerForInt(1);
```




Monomorphization

+ простая реализация

+ можно полностью использовать дженерики в коде

+ проверка типов на стадии статического анализа, компиляции и runtime

- требует дополнительной памяти

- нет типов дженериков в runtime (reflection)



Reification

До

```
<?php

class Container<T> {

    private T $data;

    public function __construct(T $data){}
}

$container = new Container<int>(1);
```

После

```
<?php

class Container<T> {

    private T $data;

    public function __construct(T $data){}
}

$container = new Container<int>(1);
```




Reification

+ нет ограничения использования дженериков в коде

+ типы дженериков есть в runtime (reflection)

+ проверка типов на стадии статического анализа, компиляции и runtime

- сложная реализация



Подходы к реализации дженериков

	type erasure	monomorphization	reification
сложность	просто	просто	сложно
синтаксис	частично	полностью	полностью
стат. анализ	+	+	+
compile checks	+	+	+
runtime checks	–	+	+
память	+	–	+
reflection	–	–	+



Проблемы реализации дженериков в PHP



Время проверки типов в PHP

100
75
50
25
0

100%
81%
80%
78%

Class
Mixed
Array
Without type



Синтаксис

```php
<?php

const FOO = 'FOO';
const BAR = 'BAR';

var_dump(new \DateTime<FOO, BAR>('now'));// дженерик
```



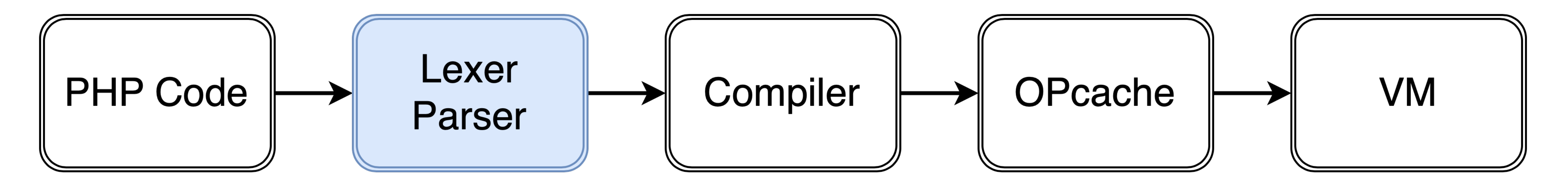

Синтаксис

```php
<?php

const FOO = 'FOO';
const BAR = 'BAR';

var_dump( (new \DateTime < FOO) , ( BAR > 'now') );// на самом деле нет
```

Синтаксис



Синтаксис (Bison)

```
line:
  %empty
|  expr { printf("%d", $1); }
;

expr:
    TOK_NUMBER    { $$ = $1; }
|   expr '+' expr { $$ = $1 + $3; }
|   expr '-' expr { $$ = $1 - $3; }
;
```




Синтаксис

Код

```
<?php

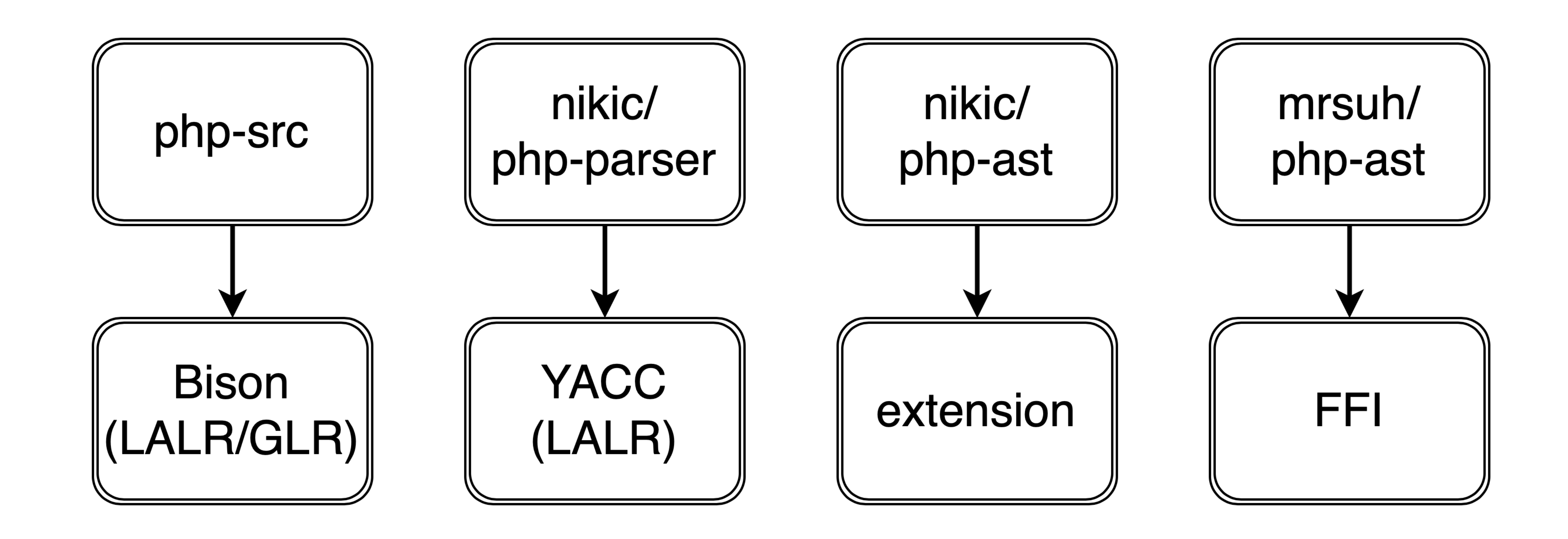
$a = 1;
```

AST

```
Expr_Assign(
    var: Expr_Variable(
        name: a
    )
    expr: Scalar_Int(
        value: 1
    )
)
```




Синтаксис (Parser)



Неготовый RFC

```php
<?php

class Traversable<T> { ... }

class A {}
class B extends A {}

if(Traversable<B> instanceof Traversable<A>) {
    echo 'ok';
}
```




Много кода для изменений





Существующие решения на PHP



Template Annotations

```php
<?php
/**
 * @template T
 */
class MyContainer {
  /** @var T */
  private $value;

  /** @param T $value */
  public function setValue($value) {}
}
```




Template Annotations

- type erasure
- не меняет синтаксис языка
- дженерики/шаблоны пишутся через аннотации
- проверки типов происходят при статическом анализе



spatie/typed

```php
<?php

$list = new Collection(T::bool());

$list[] = new Post(); // TypeError
```




spatie/typed

- не меняет синтаксис языка
- можно создать определенный список, но его нельзя указать в качестве типа переменной
- проверки типов происходят во время runtime



TimeToogo/PHP-Generics

```php
<?php

class Maybe {
    public function SetValue(__TYPE__ $value) {}
}

$maybe = new Maybe\stdClass();
```




TimeToogo/PHP-Generics

- monomorphization
- не меняет синтаксис языка
- сгенерированные классы сохраняются в ФС
- нужно использовать встроенный autoloader
- проверки типов происходят во время runtime



ircmaxell/PhpGenerics

```
<?php

class Item<T> {

    protected $item;

    public function setItem(T $item){}
}

$item = new Item<\stdClass>;
```




ircmaxell/PhpGenerics

- monomorphization
- добавлен новый синтаксис
- сгенерированные классы загружаются через eval()
- необходимо использовать встроенный autoloader
- проверки типов происходят во время runtime



Как сделать PHP-дженерики на PHP?

- добавить новый синтаксис
- найти места использования дженериков в коде
- генерировать конкретные классы на основе классов дженериков (monomorphization)
- подгружать сгенерированные классы вместо оригинальных



Новый синтаксис nikic/php-parser

```
optional_generic_params:
    /* empty */                                          { $$ = NULL; }
    |     '<' generic_params '>'                         { $$ = $2; };

generic_params:
        generic_param                                    { init($1); }
     |  generic_params ',' generic_param                 { push($1, $3); }
```




Новый синтаксис

Код

```
<?php

namespace App;

class Box<T> {

   private ?T $data = null;

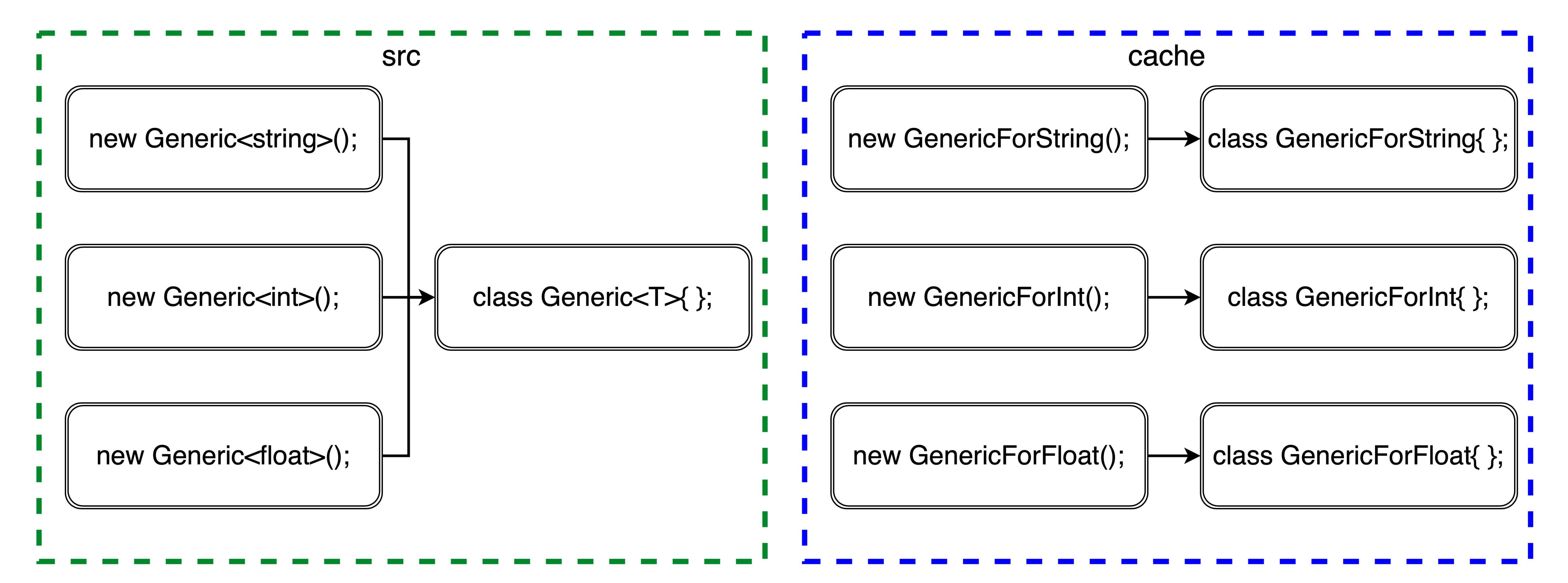
   public function set(T
$data): void {}
}
```

AST

```
…
Stmt_Class(
    name: Identifier(
        name: Box
    )
    generics: [T]
)
…
```




Генерация файлов

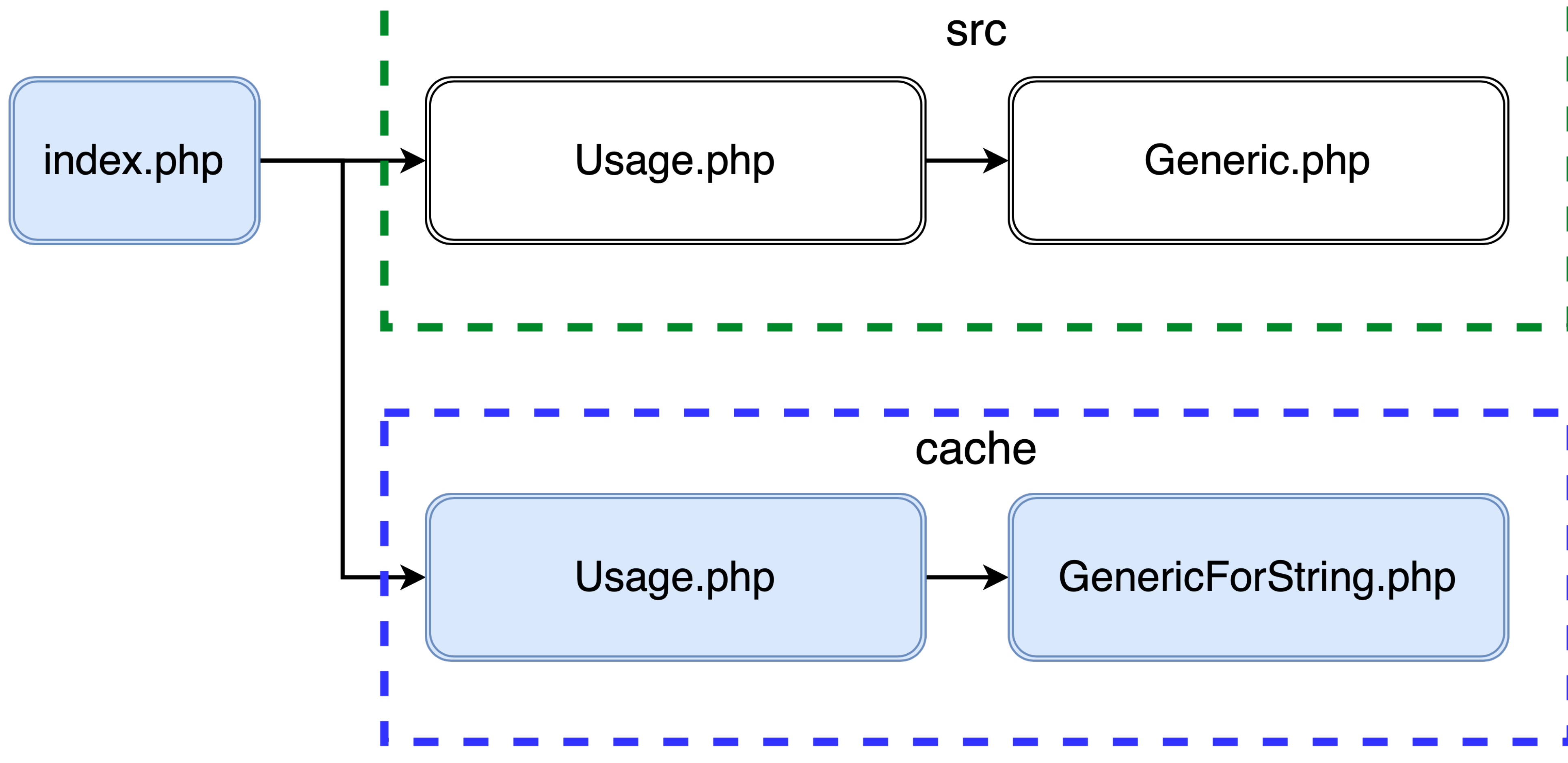


Загрузка файлов

```
{
    "autoload": {
        "psr-4": {
            "App\\": ["cache/","src/"]
        }
    }
}
```




Загрузка файлов



Загрузка файлов

index.php

```php
<?php

require_once '/vendor/autoload.php';

$usage = new App\Usage();
$usage->run();
```

cache/Usage.php

```php
<?php

namespace App;

class Usage
{
    public function run() : void
    {
        $box = new \App\BoxForString();
    }
}
```




mrsuh/php-generics

```
composer dump-generics -v
Generating concrete classes
 - App\BoxForString
 - App\Usage
Generated 2 concrete classes in 0.062 seconds, 16.000 MB memory used

composer dump-autoload
Generating autoload files
Generated autoload files

php index.php
```




Насколько быстро работает

- конкретные классы генерируются заранее, и их можно кэшировать
- чем больше конкретных классов, тем больше тратится времени на их подключение и проверку типов, а также памяти на их хранение



Нельзя использовать без composer autoload





Reflection

Типы дженериков недоступны в runtime



Поддержка синтаксиса IDE

PhpStorm

- не поддерживает синтаксис дженериков
- не имеет работающего плагина LSP
- от поддержки Hack отказались

VSCode

- есть поддержка LSP
- есть плагин для Hack



Антон Сухачёв
mrsuh6@gmail.com
github.com/mrsuh/php-generics
github.com/mrsuh/php-parser
dev.to/mrsuh
t.ly/clin

Оцените доклад



